MagiC: Software package for Multiscale Coarse-Grained Modelling. User guide. Version 2.0

Alexander Mirzoev[1,2,3], Alexander Lyubartsev[3,4]

March 8, 2016

Abstract

MagiC is a software package designed to perform systematic structure-based coarse graining of molecular models. The effective pairwise potentials between coarse-grained sites of low-resolution molecular models are constructed to reproduce structural distribution functions obtained from the modelling of the system in a high resolution (atomistic) description. The software contains tools to read atomistic trajectories generated by different simulation packages, create a coarse-grained trajectory, compute for it radial distribution functions as well as distributions of intramolecular bonds and angles, and then find effective potentials which reproduce these distributions in coarse-grained modeling. The software supports coarse-grained tabulated intramolecular bond and angle interactions, as well as tabulated non-bonded interactions between different site types in the coarse-grained system, with the treatment of long-range electrostatic forces by the Ewald summation. Two methods of effective potentials refinement are implemented: iterative Boltzmann inversion and inverse Monte Carlo, the later accounting for cross-correlations between pair interactions. MagiC uses its own Metropolis Monte Carlo sampling engine, which is efficiently parallelized providing fast convergence of the method and nearly linear scaling at parallel execution. The finally generated trajectories can be exported back to other software formats (e.g. Gromacs) for subsequent large scale simulations.

[1]School of Biological Sciences, Nanyang Technological University, Singapore.
[2]*e-mail: amirzoev@ntu.edu.sg*
[3]Division of Physical Chemistry, MMK, Stockholm University, Stockholm, SE-10691, Sweden.
[4]*e-mail: alexander.lyubartsev@mmk.su.se*

Contents

1 What's new **4**
1.1 Version 2.0 . . . 4

2 Installation and setup **4**
2.1 Getting started . . . 4
2.2 Fast Installation . . . 4
2.3 Environment variables . . . 5
2.4 Manual Compilation . . . 5
2.4.1 *CGTraj* . . . 5
2.4.2 *Magic-core* . . . 5
2.4.3 *RDF and MagicTools* . . . 6

3 Using MagiC **6**
3.1 Basic principles of the coarse-graining procedure . . . 6
3.2 *CGTRAJ*: Bead Mapping . . . 8
3.2.1 Input/output files: . . . 9
3.2.2 cgtraj.inp: main input file . . . 9
3.2.3 Example: cgtraj.inp . . . 11
3.3 *rdf.py*: Reference Distribution Functions calculation . . . 12
3.3.1 rdf.inp: main input file . . . 14
3.3.2 Example: rdf.inp . . . 16
3.4 *MagiC core*: Inverse Solver IMC/IB . . . 17
3.4.1 General description . . . 17
3.4.2 Input/Output files . . . 18
3.4.3 magic.inp: main input file . . . 19
3.4.4 Example: magic.inp . . . 22
3.5 *MagicTools*: Juggle with MagiC's data . . . 23
3.5.1 Reading the data . . . 23
3.5.2 Plotting and Inspecting the data . . . 24
3.5.3 Numerical analysis of the potentials . . . 25
3.5.4 Saving the data . . . 25
3.5.5 Exporting potentials: GROMACS . . . 25
3.6 *MagicTools* procedures reference: . . . 26
3.6.1 ReadRDF . . . 26
3.6.2 ReadPot . . . 26
3.6.3 ReadMagiC . . . 26
3.6.4 LoadDFs . . . 27
3.6.5 PlotAllDFs . . . 27
3.6.6 DFset.Plot . . . 28
3.6.7 Deviation . . . 29
3.6.8 AnalyzeIMCOuput . . . 29
3.6.9 TotalPots . . . 29
3.6.10 GetOptEpsilon . . . 30
3.6.11 PotsEpsCorrection . . . 31
3.6.12 PotsPressCorr . . . 31
3.6.13 SaveDFsAsText . . . 32
3.6.14 DumpDFs . . . 32
3.6.15 SplitDFset . . . 32

3.6.16 GromacsTopology . . . 32
3.6.17 PotsExport2Gromacs . . . 33
3.6.18 xmol2gro . . . 35
3.6.19 gro2xmol . . . 36
3.6.20 tpr2mmol . . . 36
3.6.21 Convert2NewFile_pot . . . 36
3.6.22 Convert2NewFile_rdf . . . 36
3.6.23 DFset - set of Distribution Functions . . . 36
3.6.24 DF - Distribution Function . . . 37

4 File formats **38**
4.1 .xmol . . . 38
4.2 .mmol . . . 39
4.2.1 MMOL Example: H2O.mmol . . . 39
4.3 .mcm . . . 40
4.3.1 MCM Example: DMPC.CG.mcm . . . 41
4.4 .rdf and .pot file formats . . . 43
4.4.1 Header . . . 43
4.4.2 RDF/potential record: . . . 43
4.4.3 Included Potential/RDF . . . 44

1 What's new

1.1 Version 2.0

New functionality: Possibility to fix/protect certain interaction potentials from correction, i.e. exclude a potential from update within the inverse procedure, is introduced. This gives possibility to use potentials parametrized previously in other simulations, and calculate only those potentials that yet missing in the new system.

New file formats: A new file format (compared to versions 1.*) for RDFs/potentials is introduced and old formats are depreciated (the conversion tools are provided). A new format is also introduced for MagiC-core input file, while keeping limited compatibility with the old format. The new input file format is a free text based, case insensitive and supports comment lines (starting with `!` or `#`) and empty lines. The total number of parameters was reduced and the parameters got more self-explanatory names. The old parameter names are supported as well. The RDF calculation tool rdf.py also got a revised format of the input file, which provides automatic generation of the atom pairs lists used for each specific RDF. The new rdf.py is completely written in Python (the Fortran part was overtaken by numpy), which made it faster and also removed compatibility issues with MacOS.

Improved error reporting: The error reporting was significantly improved and became (hopefully) more tolerant to the user-provided input.

2 Installation and setup

2.1 Getting started

Download and unpack the latest stable version of the software from
`http://mmkluster.fos.su.se/magic/`
or get a copy from the repository:
`hg clone https://bitbucket.org/magic-su/magic-2`
This will create folder `magic-2.x`, containing the whole software. Below we refer to MagiC as a full path to the folder at your computer.

The following software is required for the python-based part: python (usually present), numpy, matplotlib and ipython.

If you are using Ubuntu, just add these packages:
`sudo apt-get install ipython python-numpy python-matplotlib`

To compile the Fortran-based core part of MagiC, a Fortran compiler (supporting at least Fortran-95 standard) is needed. We recommend Intel Fortran or Oracle Solaris Studio, since they produce faster binaries, but GNU Fortran is also an acceptable choice. Lapack library is required. For compilation of the program for parallel execution, MPI library is required.

2.2 Fast Installation

Run `install.sh` script. That is it!

By default, fast installation script uses GNU Fortran compiler and compiles MagiC-core in serial (non-MPI) version. If you want to specify different compiler/options you need to uncomment corresponding lines in the installation script:

Open file install.sh for editing. Find **install CGTraj** and **install MagiC** sections and uncomment the lines with `make` option corresponding to your Fortran compiler and MPI-library.

Run the script. It will result in compilation of subparts of the package, linking executable in folder `$MAGIC/bin` and exporting `PATH`, `LD_LIBRARY_PATH` and `PYTHONPATH` environment variables. If you face errors during execution of install.sh, you can try to compile MagiC manually step by step as described in the Manual compilation section below.

2.3 Environment variables

The simplest way to setup the necessary environmental variables and get access to the compiled executables and libraries is to use the script `setvars`:
`source setvars.sh` (if you use `bash`)
or
`source setvars.csh` (if you use `tcsh`)

Otherwise you can set the values in startup files of your shell such as `$HOME/.bashrc` in case of interactive session or `.profile` in case of remote login session, or `.tcshrc` if you use `tcsh`

For `bash`, add the following lines (specify the location of MagiC in the first line):

```
# Define location of MAGIC below.
MAGIC=<PATH_TO_LOCATION_OF_MAGIC>
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MAGIC/MagicTools/xdrfile-1.1.1
export PYTHONPATH=$MAGIC/MagicTools/xdrfile-1.1.1/src/python:$PYTHONPATH
export PYTHONPATH=$MAGIC/MagicTools:$MAGIC/MagicTools/lib:$PYTHONPATH
export PATH=$PATH:$MAGIC/bin
```

If you use `tcsh`, set up the same environmental variables using `setenv` directive.

2.4 Manual Compilation

2.4.1 *CGTraj*

Enter folder `CGTraj-1.3` and run `make` specifying the compiler as an argument to `make`, e.g. `make intel` for Intel Fortran, `make oracle` for Oracle Solaris studio, or `make gfortran` for GNU Fortran (default option), or edit one of the available `Makefile`s. If compilation went successfully you get an executable file called `cgtraj`.

2.4.2 *Magic-core*

To compile this part of the package, you need a Fortran compiler and LAPACK linear algebra library. Lapack is included in Intel MKL which comes with Intel Fortran and it is also included in Oracle Solaris Performance Library, which

comes with Oracle Solaris Studio. Other option is GNU Fortran (gfortran) and a repository build of Lapack (liblapack-dev). If you have any of them just use corresponding Makefile for compilation or use `make` with argument, for example `make intel`. Run `make` to see all options available.

For better performance it is highly recommended to compile and run MagiC in the parallel mode. To do so you need a MPI-library installed. Examples of open-source MPI libraries for which MagiC was tested are OpenMPI, and Mpich. For compilation of MagiC in the parallel mode, use one of Makefiles `Makefile.gfortran-mpi`, `Makefile.intel-mpi` or `Makefile.oracle-mpi`, or adopt one of these files to the specific combination of Fortran compiler and MPI-environment on your computer.

The result of successful compilation is a binary file `magic` with possible extentions describing compilation conditions, which you can copy/link to your default bin folder.

2.4.3 *RDF and MagicTools*

These parts of the package are written in Python and do not require explicit compilation, you need only to link file `rdf.py` to your default `bin` folder and add `MagicTools` and `MagicTools/lib` folder to your `PYTHONPATH` environment variable.

To check if the library is added successfully, open terminal, run `ipython` and load the library:
`import MagicTools`
If no error message appeared, the RDF and MagicTools modules are correctly connected.

For reading binary CG trajectories produced by GROMACS (.xtc and .trr) the RDF utility relies on xdrfile library, which need to be compiled. To compile it, enter `MagicTools/xdrfile-1.1.1` folder and run
`./configure --enable-shared`
You can also specify exact location for the compiled library files by adding
`--prefix=/path/to/library/folder`
to the `configure` arguments. Build the library by `make install`. If you have specified non-default location for the library, add it to the LD_LIBRARY_PATH environment variable.

Finally, add python wrapper for the library in PYTHONPATH:
`export PYTHONPATH=$MAGIC/MagicTools/xdrfile-1.1.1/src/python:$PYTHONPATH`

3 Using MagiC

3.1 Basic principles of the coarse-graining procedure

MagiC is a software package designed to perform tasks of multiscale structure based coarse graining in molecular simulations. This is done by extracting radial distribution functions (RDF) and other structural information from a high resolution (fine grained) simulation of the system of interest (reference system), and then to generate effective potentials which reproduce these RDFs in a coarse-grain model by means of the inverse Monte-Carlo method or Iterative Boltzmann inversion method. Such potentials can be further used for large scale simulation.

In general systematic coarse graining can be considered as a multi-stage process which leads from a high resolution system description to the low resolution one (see figure 1).

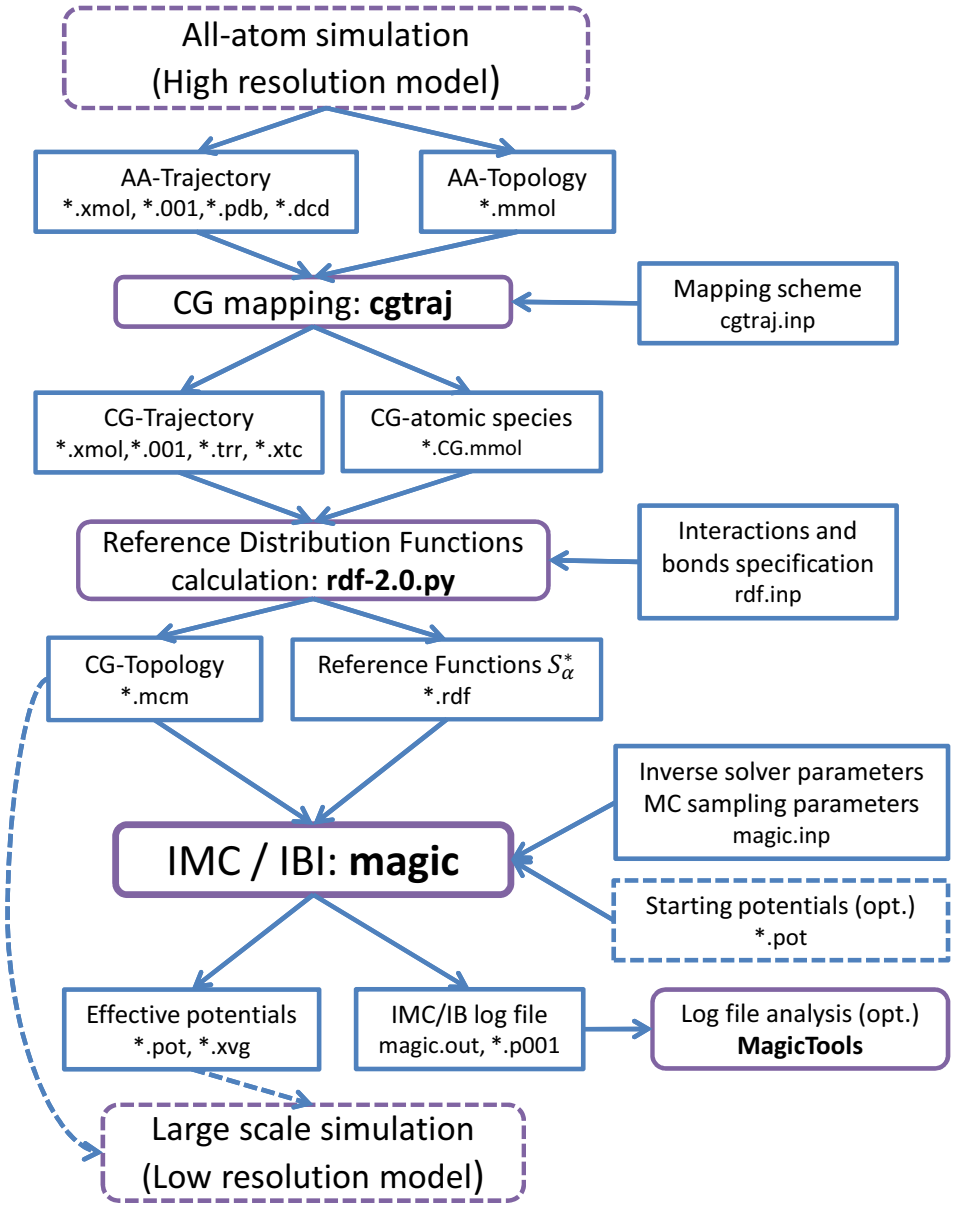

Figure 1: Systematic Coarse-Graining with MagiC: General outline. Blue rectangles denote input/output data; purple rectangles denote data processing procedures. Optional input data and use of external software are marked with dashed frame.

Each step (shown in purple) uses results of the preceding stage output as an input (input/output is shown in blue), and additional input provided by user (rightmost blue blocks).

Five stages can be distinguished:

1. The system of interest is simulated at high resolution, e.g. using Molecular Dynamics simulation with all-atom (AA) force field. Such a simulation results in atomistic (AA) trajectory which is supposed to sample the atomistic system well enough. This simulation can be performed by any suitable external molecular dynamics (or Monte Carlo) software.

2. A coarse-grained (CG) trajectory is generated from the atomistic trajectory obtaned during the first stage. This is performed by utility *cgtraj* which is a part of MagiC. It converts AA-trajectory into CG-trajectory, using a user provided mapping scheme which states the correspondence between atomistic and CG representations for every molecular type. This stage results in a coarse-grained trajectory and mass/charge properties of CG-beads stored in molecular description files (`.mmol`).

3. Structural reference distribution functions are calculated by utility *rdf.py*. Since every distribution function corresponds to a specific interaction, definitions which interactions and bonds are present in the CG system are defined at this stage.

4. The inverse problem is solved by the Inverse Monte Carlo or Iterative Boltzmann Inversion methods. This is the key stage, which is done by a core of the package which is called *magic core*. During this stage, effective potentials between CG sites are iteratively refined to fit the RDFs obtained in atomistic simulations. Also, an extended log-file is generted which reports details of every IMC/IBI iteration. The files can be analyzed by a set of post-processing tools *MagicTools*, which allow to plot the convergence rate, effective potentials from each iteration, potential corrections at each iteration, intermediate RDFs, etc.

5. Once the effective potentials reproducing the reference RDFs with required precision are obtained, they can be exported by *MagicTools* to an external MD software and used for further simulations of the large scale CG system. At the moment export to GROMACS format is provided, extentions to other mesoscale simulation software accepting tabulated potentuals can be made relatively straightforwardly.

Since MagiC is implemented as a set of separate programs, it is possible to perform different tasks on different computers, for example run the most time-consuming part of the calculations, inversion of RDFs (stage 4), on a high performance cluster, and perform analysis on a local desktop computer.

3.2 *CGTRAJ*: Bead Mapping

This is a tool to map/convert a high resolution (all-atom) trajectory to a coarse-grained trajectory.

3.2.1 Input/output files:

Input files:

***Trajectory** For reading trajectories, MagiC uses trajectory reading interface inhereted from MDynaMix molecular dynamics package. The following formats are supported:

XMOL - Text-based XYZ trajectory format *.xmol.

PDB - Text-based trajectory format *.pdb

MDYN - MDynaMix trajectory binary format. It is usually given as a set of numerated files *.001,*.002, etc.

DCD - CHARMM/NAMD binary trajectory file format: *.dcd

Each of these types of trajectory can be presented as a set of files with extentions *.001,*.002,*.003,...

It is supposed that in all cases, the atoms and molecules are arranged in the following way:

`<molecules of type 1><molecules of type 2>...`

in each molecule type:

`<molecule 1><molecules 2>...`

in each molecule:

`<atom1><atom2>...` (the same atom order must be in all molecules of this type)

***Molecular type** descriptions for each type involved: *.mmol files (optional, see detailed description below).

***Main input** providing the mapping scheme, input trajectory and output files parameters.

Output files:

CG trajectory in XMOL format.

CG molecular types descriptions in .mmol format, but without bonds: *.CG.mmol . They will be used at the next stage during RDFs calculations.

Run: Usage: `> cgtraj cgtraj.inp`

3.2.2 cgtraj.inp: main input file

The file consists of two independent parts. The first part describes input atomistic trajectory, and the second part describes CG-bead mapping scheme.

Trajectory reading subroutine was inherited from *tranal* utility of MDynaMix, and it has the same syntax. The first part of the input file is written in Fortran NAMELIST format which looks like:

```
$TRAJ
parameter=value(s),
...
$END
```

"TRAJ" is the name of this NAMELIST section. The following parameters shall be defined:

`NFORM = <format>`

where `<format>` is one of:

- `MDYN` - MDynaMix binary trajectory (default)
- `XMOL` - XMOL trajectory. It is implied, that the commentary (second) line of each configuration is written in the format:

  `(char) <time> (char-s) BOX: <box_x> <box_y> <box_z>`

  where `(char)` is any character word, `<time>` is time in fs,
  `<box_x> <box_y> <box_z>` (following after keyword `BOX`) are the box sizes.
- `PDBT` - PDB trajectory as generated by "trajconv" utility of GROMACS simulation package.
- `DCDT` - DCD trajectories generated by NAMD package

`FNAME = <file_name>`

set the base name of the trajectory files. The trajectory must be written as a sequence of files `<file_name>.001` , `<file_name>.002` and so on, the largest possible number being `<file_name>.9999` .

`PATHDB = <value>`

Directory with molecular description files (.mmol). Default is the current directory (.) .

`NTYPES = <value>`

Number of molecule types in the trajectory

`NAMOL = <name1> [,<name2>,...]`

`NTYPES` names of molecules. It is supposed that files `<name1>.mmol`, `<name2>.mmol`,... describing the molecules are present in the directory defined by `PATHDB`. Format of .mmol files is the same as for MDynaMix program. For analyzing trajectories generated by other programs, .mmol files are still used to provide information about atomic masses and charges. It is however enough to have only the first section of .mmol files containing description of atoms.

The program may work without .mmol files, if parameters `NSPEC` and `NSITS` (see below) are given. In this case, the masses of all atoms are set to 1 and the charges to 0, which will result in definition of CG sites as geometric centers of the atomic groups, and zero charges of CG sites (the later can be manually corrected at the next stage).

`NSPEC = <n1>[,<n2>,...]`

Number of molecules of each type (`NTYPES` numbers). This parameter is not necessary in MDynaMix binary trajectories.

`NSITS = <n1>[,<n2>,...]`

Number of atoms on each molecule of each type (`NTYPES` numbers). This parameter is not necessary if .mmol files for each molecular type are provided.

`NFBEG = <value>`

Number of the first trajectory file (integer between 0 and 9999)

`NFEND = <value>`

Number of the last trajectory file (integer between 0 and 9999)

`IPRINT = <value>`

Defines how much you see in the intermediate output. The final output with analysis of results does not depend on it. Default value is 5.

`BOXL = <x-box-size>`

`BOYL = <y-box-size>`

`BOZL = <z-box-size>`

define the box size if it is not present in the trajectory (implies constant-volume simulation) If information of box sizes is present in the trajectory, box size parameters from the input file are ignored.

`ISTEP = <value>`

Specifies that only each `ISTEP`-th configuration from the trajectory is taken for the analysis. Default is 1.

The second part of the main input file for `cgtraj` utility, which describes CG bead mapping scheme, has a hypertext-like format. The section starts with keyword: `BeadMapping` and ends with `EndBeadMapping`. Each coarse grain molecular type is described in a separate section, which starts with tag `CGMolecularType: <CGMolecularTypeName>` and ends with `EndCGMolecularType`. Inside each section, parental molecular type name and CG beads definition should be given. The parent's name is defined by the tag

`ParentType: <ParentMolecularTypeName>`.

CG beads are defined in a line-per-bead way, where every line has the following structure:
`<Bead name>:<Number of atoms in the bead>:<list of atoms atom1,atom2,...>`,

where list of atoms is a comma separated list of atom numbers according to the *mmol*-file describing parental molecular type. A file named `<CGMolecularTypeName>.CG.mmol` containing description of a coarse-grained molecule will be created for every defined CG-molecular type.

The keywords/tags are not case sensitive, and spaces will be automatically removed from the text.

3.2.3 Example: cgtraj.inp

This is an example of the input file, which reads a trajectory written in binary MDynaMix format stored in 10 files: `dmpc16.001, dmpc16.002,...,dmpc16.010`.

The system presented in the trajectory consists of 16 DMPC lipid molecules dissolved in 1600 water molecules. The bead mapping scheme is shown on figure 2. The resulting CG DMPC have 10 beads. In the example below each water molecule will be represented by a single bead of "CG water". In order to completely remove the water and make implicit solvent model, the line beginning with H2O: should be commented out.

```
 &TRAJ
 NFORM='MDYN',
 FNAME='./MDynamix/dmpc16',
 PATHDB='./',
 NTYPES=2,
 NAMOL='dmpc_NM','H2O',
 NSPEC=16,1600,
 NFBEG=1,
 NFEND=10,
 ISTEP=1,
 IPRINT=6,
 &END


 BeadMapping
 CGTrajectoryOutputFile:cgtraj.001
  CGMolecularType:dmpc_NM.CG
    ParentType: dmpc_NM
    N:16:43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58
    P:11:59,60,61,62,63,64,65,66,67,68,69
    C1:9:1,2,3,4,5,73,74,75,76
    C2:12:6,7,8,9,10,11,12,13,14,15,16,17
    C3:12:18,19,20,21,22,23,24,25,26,27,28,29
    C4:13:30,31,32,33,34,35,36,37,38,39,40,41,42
    C5:8:70,71,72,77,78,79,80,81
    C6:12:82,83,84,85,86,87,88,89,90,91,92,93
    C7:12:94,95,96,97,98,99,100,101,102,103,104,105
    C8:13:106,107,108,109,110,111,112,113,114,115,116,117,118
  EndCGMolecularType
  CGMolecularType:H2O.CG
    parenttype:H2O
    #comment the line below to exclude the water (make implicit solvent model)
    H2O:3:1, 2, 3
  endcgmoleculartype
EndBeadMapping
```

3.3 *rdf.py*: Reference Distribution Functions calculation

This section describes calculation of the structural distribution functions (DF), which will be used as a reference for the effective potentials calculation during the inverse process. Note, that after the previous stage (cgtraj), the generated CG trajectory does not have any information about chemical CG types

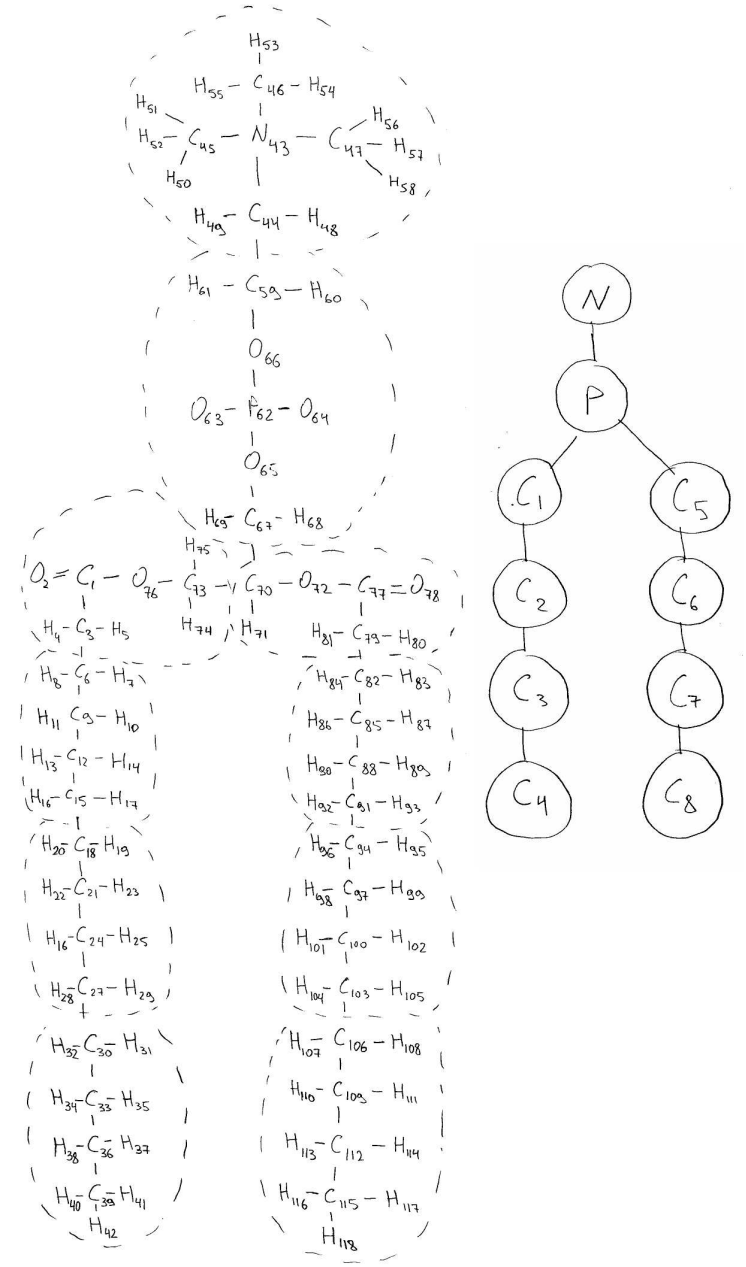

Figure 2: Simple mapping scheme of DMPC phospholipid, which consists of 118 atoms, into 10-beads CG model. The water is mapped into a single bead.

or bonding, and this information is provided at this stage when CG types and their bonding is defined. Each distribution function will result in an individual interaction potential, therefore definition of groups of beads, which belong to a

certain DF, is equivalent to definition of specific interactions in the system.

Input files:

***Molecular type description** for each type present in the system: **.mmol*.

***CG trajectory** : **.xmol*.

***Main input** defining RDF calculation parameters, input trajectory, CG-atom/bead types and the list of RDFs to calculate (and beads included in each specific RDF): *rdf.inp*
NB! The new input format was introduced in MagiC 2.0, however, the old format of v.1.x is still supported and detected automatically.

Output files:

Reference distribution functions : **.rdf*. See format details

Coarse-Grain topology **.mcm*. See format details.

Run:

```
rdf.py -i rdf.inp
```

3.3.1 rdf.inp: main input file

RDF input file consists of several parts: RDF-calculation parameters (`&Parameters`), definition of Coarse-Grain atom types (`&CGTypes`), and the list of RDFs to calculate (`&RDFsNB, &RDFsB, &RDFsA`).

RDF calculation parameters: (`&Parameters ... &EndParameters`)
The section describes the input trajectory and defines resolution and cut-off ranges for the reference distribution functions. Note that at this point the trajectory should be already coarse-grained and solvent removed (if the later supposed to be implicit).

The following parameters shall be specified. Old names of the parameters used in MagiC v.1 are stated in parentheses:

`OutputFile = <filename>` The name of the output file containing a set of calculated RDFs (FOUTRDF)

`TrajFile = <filename>` The name of the CG-trajectory file. (FNAME). The file format will be detected from the extension, or can be stated explicitly in the parameter `NFORM`

`NFORM = <format>` (Optional) Explicitly specified format of the CG-trajectory. Can be XMOL, TRR or XTC. Detected automatically from the trajectory file extension.

`BeginFile=<value>, EndFile=<value>` (Optional) If the trajectory is split into a number of files emumetated by file name extentions (.001, .002, .003, ...), these parameters specify a range of the files to read. (NFBEG, NFEND)

`Step = <value>` (Optional) How often to read frames from the trajectory (ISTEP). Default: 1

`NMType = <value>` Number of molecular types present in the CG-trajectory. (NTYPES)

`NameMType = Type1, Type2, ... , Type(NMType)` Names of molecular types present in the system. Each type should have a molecular description file, having the same name as the molecular type (NAMOL)

`NMolMType = Num1, Num2, ... ,Num(NMType)` Number of molecules of each molecular type present in the system (NSPEC)

`RMaxNB=<value>` Cut-off distance for intermolecular / non-bonded RDFs (RDFCUT)

`RMaxB=<value>` Cut-off distance for intramolecular RDFs (RMAX)

`ResolNB=<value>` Resolution (Å) of the histogram for intermolecular RDF calculation (DELTAR)

`ResolB=<value>` Resolution (Å) of the histogram for intramolecular RDF calculation (DELTARI)

`ResolA=<value>` Resolution (degrees) of the histogram for intramolecular angular distrubution functions calculation (DELTAPHI)

`Box= <X> <Y> <Z>` Size of the periodic box (Å) used as PBC. Optional. Used only if the periodic cell size was not specified in the trajectory file.

Bead types: `(&CGTypes, ... , &EndCGTypes)`
Here bead types (CG-atom types) are introduced and beads belonging to each type are specified. This is done by a list of lines having a format `<Name of CG-type>:<NameBead1 NameBead2 NameBead3>`, one line per each type, bead names are space separated. Note, that the order of bead type lines will define indexes of the bead types set in the `.mcm` -files.

Non-Bonded RDFs `(&RDFsNB, ... , &EndRDFsNB)`
Here we define a list of reference distribution functions for non-bonded interactions, which are radial distribution functions. For each function a list of bead-pairs (CG atom pairs) involved in the specific interaction shall be provided. It is possible to generate the list automatically between all or some of pairs of bead types using the following commands:

`add: all`
This will generate automatically a list of RDFs which includes all possible RDFs based on pair combinations of CG-atom types. For each pair of CG-atom types a RDF will be determined, which includes all pairs of CG atoms of the specified types, and effective potential for this pair of atom types will be calculated on the next stage. With this option, all possible NB-RDFs will be taken into account. This is the most common regime.

`add: <CGType> -- <CGType>`
Create a list of CG-atom pairs having the given CG-atom types, and include it into calculation of RDFs. This will add a single RDF to the list.

`add: <CGType1> -- <CGType2>: AName1 AName2, AName3 AName4`
Explicitly add pairs of atoms AName1-AName2, AName3-AName4 to the RDF for the given pair of CG-atom types. This is the most specific way of setting the atom-pairs list for a given RDF.

`del: <CGType> -- <CGType>`
Remove a specific RDF (interaction) from the set of RDFs generated up to this line.

`del: <CGType> -- <CGType>: AName1 AName2, AName3 AName4`
Exclude a specific pair of atoms from the RDF for given atom types.

RDFs for Pairwise Bonds `(&RDFsB, ... , &EndRDFsB)`

In this section reference distributions for pairwise bonds (e.g. bond length distributions) will be specified. Note that this is important information which determines bonding in the CG molecule, which has to be specified explicitly.

Each bond should be specified by the following line: For every independent bond type the molecular type it belongs to should be specified, with the relative index of the bond and atom pairs involved in the bond.

`add: <MolType>: <BondIndex>: <AName1> <AName2>, <AName3> <AName4>`
where `<MolType>` is index of the molecular type, `<BondIndex>` is the index of the bond in the given molecular type, and pairs `<AName1> <AName2>, <AName3> <AName4>,...` determine CG atoms within the molecule connected by the given bond type.

RDFs for Angle-bending bonds `(&RDFsA, ... , &EndRDFsA)`
In this section reference distributions for angle-bending bonds (e.g. bond angle distribution) are determined. It can be done manually, similar to specifying pairwise bonds, or deduced automatically by setting an A-bond between every two interconnected pairwise bonds (excluding cases when the end atoms of the angle are already connected by a bond). Note that pairwise bonds shall be set in prior, i.e &RDFsA-section shall go after &RDFsB-section. The following keywords can be used in this section:

`add: all`
Automatically deduce angle-bending bonds for all molecular types of the system

`add: MolType : all`
Automatically deduce angle-bending bonds in the given molecular type

`add: <MolType>: <BondIndex>: <AName1> <AName2> <AName3>, ..., ...`
Explicitly add triplet (triplets) of atoms to the given angle-bending bond of the given molecular type

`del: MolType : all`
Discard all angle-bending bonds in the given molecular type

`del: MolType : <BondIndex>`
Discard given A-bond

`del: MolType : <BondIndex>: <AName1> <AName2> <AName3>, ...,`
Remove given atoms from the defined previously A-bond

Note that every pair of atom involved in a bond (pairwise or angle-bending) is automatically excluded from non-bonded RDF calculation and from non-bonded interactions.

3.3.2 Example: rdf.inp

This is an example of rdf.inp file which defines reference distribution function calculation for a interactions shown on figure 3.

```
&Parameters
 TrajFile = cgtraj.dmpc16-400ns.xmol
 NMType = 1
 NameMType = dmpc_NM.CG.mmol
 NMolMType = 16
 OutputFile = dmpc16-100aa.rdf
 RMaxNB = 20.
 RMaxB =10.0
 ResolNB =0.1
 ResolB=0.02
 ResolA=1.0
&ENDParameters

&CGTypes
N:N
P:P
CH:C2 C3 C4 C6 C7 C8
CO:C1 C5
&EndCGTypes

&RDFsNB
Add: all
# Add: N--P
# Add: N--P: N P
# Add: N--CO: N C1, N C5
&EndRDFsNB

&RDFsB
add: dmpc_NM.CG: 1: N P
add: dmpc_NM.CG: 2: P C1, P C5
add: dmpc_NM.CG: 3: C2 C3, C3 C4, C6 C7, C7 C8
add: dmpc_NM.CG: 4: C1 C2, C5 C6
add: dmpc_NM.CG: 5: C1 C5
&EndRDFsB

&RDFsA
Add: all
#add: dmpc_NM.CG: All #Other way - generate all A-bonds for the molecule
#add: dmpc_NM.CG: 6: N P C1 #Example - create N-P-CO A-bond
&EndRDFsA
```

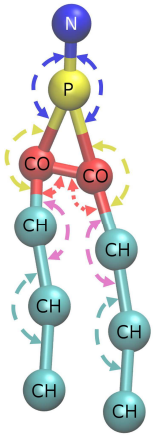

Figure 3: Example: 10-beads CG model of DMPC-lipid. Beads and bonds of same color have same type; solid lines denote covalent bonds; dashed arrows denote angle bending bonds.

3.4 *MagiC core*: Inverse Solver IMC/IB

3.4.1 General description

This is the main module of the whole software package. It performs Metropolis Monte-Carlo sampling of the system described by some trial set of potentials, then compares sampled distribution functions with the reference ones, and introduces a correction to the set of potential. Then the new iteration starts, with the corrected set of potentials. The process repeats the specified number of iterations. The process of inversion of RDFs can be regarded as completed when agreement between sampled and reference distribution functions is reached.

The software automatically analyses the provided input files: molecular descriptions, RDFs and/or potentials files and checks them for consistency. If potentials for some interactions are not provided, they will be deduced from the corresponding RDFs. By default a zero-potential (except the core with zero reference RDF) is used as a starting potential for all NB-interactions, while potential of mean force is used for the bonding interactions (which can be of type B or A). It is also possible to use potential of mean force as a starting potential for non-bonded interactions (not always suitable for multisite coarse-grained molecules). A user can specify the kind of trial potentials for every interaction in the `.rdf` file using keywords `&InitZero, &InitPMF`

If neither potentials nor RDFs are provided for certain non-bonded interaction, then such interaction will not be used in the MC simulation (which is equivalent that corresponding interaction potential is set to zero). In case if RDF-file is absent, no inverse procedure will be performed and the program will just run a standard MC simulation with the supplied potential. In addition it is also possible to fix some potentials and do not update them in the inverse procedure using keyword `&Fixed` in the potential file.

3.4.2 Input/Output files

Input files:

Main input specifying parameters of the Monte-Carlo sampling, inverse solver and input/output files: *magic.inp*

CG topology for every CG-molecular type: **.mcm*

Reference distribution functions : **.rdf*

Starting trial potentials **.pot*. If not provided (or partially provided), the missing trial potential will be deduced from the RDF-file as potential of mean force or zero-potential, depending on user's choice.

Initial geometry for MC-process in xmol-format (optional). In case of parallel execution, an individual file should be provided for each parallel process:
name-of-the-system.p<process-number>.i<iteration-number>.start.xmol

Output files:

General output By default it is printed on the screen, but it is recommended to redirect it to a file (see example below)

Log/journal for every parallel process . name-of-the-system.p<process-number>. In serial run it is written to the general output.

Resulting effective potentials (starting potential for current iteration + correction): *name-of-the-system.i<iteration-number>.pot*

Monte-Carlo trajectory of each parallel process:
name-of-the-system.p<process-number>.xmol

The final snapshot of the system of each parallel process and iteration:
name-of-the-system.i<iteration-number>.p<process-number>.start.xmol This file can be used as a starting configuration for a consequent MC run. In case of a parallel MC run, a set of files is produced which are used as starting configurations for each processors at the next iteration.

Execution:

serial execution : `magic magic.inp`

parallel execution : `mpirun -np number_of_processes magic magic.inp >magic.out`

3.4.3 magic.inp: main input file

In the MagiC-core input file (version ¿ 2.0) parameters are specified as a set of keywords
`ParameterName = Value`
and have self-explanatory names (old names from version 1.x are also supported). Warnings or error messages are issued for missing compulsory parameters or inconsistent values. Lists of variables (vectors) should be comma separated (i.e. BOX = X, Y, Z). Comment lines denoted by or # are supported as well as empty lines. For logical parameters most possible acronyms, such as F, .F., False, FALSE are accepted.

For convenience the parameters are divided into five groups. There is however no need to follow this order, the program accept parameters in any order

they follow. The old names of parameters (which is still possible to use) are given in the brackets.

The following parameters can be defined. In the second brackets an alternative keyword corresponding to v. 1.x is given. Keywords marked with * are mandatory.

System parameters:

NMType* [NTYP] Number of different molecule types (species) present in the system.

NameMType* [NAMOL] Names of the molecule types present in the system, separated by comma. Every molecule type should have a respective description file(.mcm). For example `NameMolType = H2O, DMPC` defines 2 names: `H2O`-for the first molecule type, and `DMPC` for the second one. The description files should be named `H2O.mcm` and `DMPC.mcm`.

NMolMType* [NSPEC] Number of molecules of each type, written as a comma-separated list, e.g. `NMolMType=392,3` defines system, which consists of 392 molecules of the first type, and 3 molecules of the second type.

LMoveMType [LMOVE] Which molecular types are allowed to move in the Monte Carlo simulation. List of comma-separated logical values. Default `LMOVE = True,.., True` , i.e. all molecules are allowed to move. Frozen molecules coordinates has to be specified in a *.xmol file given in `InputFrozenCoords` parameter.

Epsilon [EPS] Dielectric permittivity constant defining electrostatic interactions in the system. Default: 1.0

TEMP* Temperature of the system, K

Box* [BOXL,BOYL,BOZL] Periodic cell dimensions in Å, separated by comma. The software uses rectangular periodic boundary conditions.

Monte Carlo parameters:

MCSteps* [NMKS] Total number of Monte Carlo steps to be performed on every iteration (including equilibration).

MCStepsEquil* [NMKS0] Number of Monte Carlo steps to be performed for the equilibration.

MCStepAtom* [DR] Maximum displacement in a Monte Carlo single atom displacement step, Å. Default: 1.0

MCStepTransMol [MCTRANSSTEP] Maximum displacement in a MC translation of a whole molecule, Å. Default: 0.0

MCStepRotMol [MCROTSTEP] Maximum degree of MC rotation of the whole molecule, deg. Default 0.0

iMCStepTransMol [ITRANS] How often (in terms of MC steps) to perform random translation of a randomly chosen molecule. Default: 0, i.e. never

iMCStepRotMol [IROT] How often (in terms of MC step) to perform random rotation of a randomly chosen molecule. Default: 0, i.e. never

iCalcEnergy* [IOUT] How often to recalculate the total energy and write energies and pressure to the log-file. If the difference in total energy before and after the recalculation is larger than $0.01 k_B T$, a warning message will be given. This procedure is computationaly expensive (order N^2), and should not be performed too often.

RCutEl* [RECUT] Real space cutoff for electrostatic energy in real-space part of the Ewald sum. It is recommended to set it equal to cutoff radius of RDFs and short-range interactions, but in some cases other choices can be reasonable.

AF, FQ Ewald summation parameters: The electrostatic energy in Ewald method can be expressed as

$$U_{el} = \frac{1}{2V} \sum_{k \neq 0}^{k^2 < k_{cut}^2} \frac{4\pi}{k^2} |\rho(k)|^2 \exp\left(-\frac{k^2}{4\alpha^2}\right) - \frac{\alpha}{\sqrt{\pi}} \sum_{i=1}^{N} q_i^2 + \frac{1}{2} \sum_{\substack{i \neq j \\ r_{ij} < r_{cut}}}^{N} \frac{q_i q_j \, erfc\left(\alpha r_{ij}\right)}{r_{ij}} \tag{1}$$

where, $\alpha = \frac{AF}{r_{cut}}$, and $k_{cut}^2 = 4\alpha^2 \, FQ$. In other words, the precision or the first sum is defined by $\exp(FQ)$, while accuracy of the third sum is defined by $erfc(AF)$. Default: `AF=3., FQ=9.0`

RandomSeed [NRS] Initial seed for the random number generator.

KeepStructure [LCRDPass] Define if the final structure of the previous inverse iteration shall be used as starting for the consequent iteration. Otherwise the starting configuration will be generated randomly. Default: False

Inverse procedure parameters:

UseIMC [LIMC] Inverse solver selection. If true, the Inverse Monte Carlo method is used, otherwise iterative Boltzmann inversion is used. Default: True (IMC)

NIter* [IREPT] Number of inverse iterations to perform. Default: 1

RegP [REGP] Regularization parameter for potential correction. This parameter defines the relative weight of correction, and has a value between 0 and 1. In case of instability (each next iteration returns larger deviation from reference RDF), value of REGP should be decreased. It is also advisable if correction to potential at each iteration exceeds treshold value (default 2 $k_B T$). REGP can be again increased closer to 1 if the iteration process is stable and correction to the potential at each iteration is small. Default: 1.0

iAverage* [IAV] How often to compute averages over the system. Since computation of the averages (RDFs and cross-correlations) involves calculation of distances between all pairs of atom, this procedure is rather expensive, and should not be performed too often. The recommended value is of the order of number of CG atoms in the system The averaging starts after the equilibration, i.e. when first MCStepsEquil steps have passed.

MaxPotCor [DPOTM] Maximal change of potential value at every point during correction procedure, given in $k_B T$ units. Default: 2.0

MaxRelDif [RTM] Parameter limiting maximum relative difference between reference and resulting averages. Default: 10.0

iPotCorrCheck How often to perform potential correction check. The program gathers accumulated statistics from all the processes, and then calculates sampled distribution functions and potential corrections. However, this corrections are not applied to the actual interaction potential, but just printed to the log file. This allows user to analyze how well both distribution functions and potential corrections are converged after given number of MC steps of an inverse iteration. The checks are performed after equilibration. Default 0, i.e. no check at all.

ProhibPotLevel [POTCUT] Prohibiting potential level. Relatively high value of potential in kJ/mol to define a core region of the potentials at distances where the corresponding RDFs are zero, to avoid MC steps leading to such distances. Default: 1000.

Input-Output parameters:

Output* [BASEOUTFILENAME] Prefix name of the system (filename template) to use for writing output files. All names of output files will begin with the given prefix.

VerboseLevel [IPRINT] Verbosity level of the log-file. 1-minimum level, 10 - maximum level. Default: 5.

WriteTraj [ITR] How often (in terms of MC steps) to write current geometry to the trajectory file. Default: 0 - do not write it at all.

InputRDF [FILRDF] Input file with reference distribution functions. Required for inverse procedure, otherwise only a direct MC simulation will be performed

InputPotential [FILPOT] Input file with a set of trial potentials.

InputStartCoords [FSTART] Name of the input file (or prefix for a set of files) with starting coordinates, excluding 'p001.start.xmol' suffix. If not provided, the starting geometry will be randomly generated.

InputFrozenCoords [FCRD] Name of a single *.xmol file with starting coordinates of all frozen molecules in the system. It will be used to define starting location of the frozen species for every inverse iteration. The file is almost the same as start.xmol, but it does not contains moving molecules/atoms.

DumpLastConf [LXMOL] If true, program dumps the last configuration of MC process in file (or set of files) with ".start.xmol" extention. It is done after every inverse iteration on every parallel process. Default: False - do not dump. In case of parallel execution, output filenames have extentions `<Output>.i<iteration>.p<process>.start.xmol`

3.4.4 Example: magic.inp

Here is an example of the input file, representing the system of 2 molecule types (MT1 and MT2), having 10 molecules of each. The program shall read RDF from file MT1MT2.rdf and the IMC will be used for inversion, making

10 iterations in total. As the potentials are not provided, the trial potentials will be deduced from RDFs according to the settings in the RDF file 5 millions MC steps are to be made at every inverse iteration, and half of them are for equilibration.

```
NMType = 2
NameMType = MT1, MT2
NMolMType = 10, 10
LMoveMType = TRUE, T
Box = 15.0, 15.0, 15.0
Epsilon = 1.0
 TEMP=303.

 MCSteps = 5000000,
 MCStepsEquil = 2500000,
 MCStepAtom = 0.2
 MCStepTransMol = 1.0
 MCStepRotMol = 0.2
 iMCStepTransMol = 50
 iMCStepRotMol = 50
 iCalcEnergy = 100
 RCutEl=0.7
 AF = 2.6
 FQ = 8.0
 ProhibPotLevel=1000.0
 RandomSeed=51

 UseIMC = True
 NPointsNB = 70
 NIter=10
 IAverage=50
 REGP = 0.1,
 MaxPotCor=2.0
 KeepStructure=False
 MaxRelDif=10.0
 iPotCorCheck = 250000

 VerboseLevel=5
 InputRDF= MT1MT2.rdf

 Output = 01.111.MT1MT2
 DumpLastConf = .false.
 WriteTraj = 100000
```

3.5 *MagicTools*: Juggle with MagiC's data

This part of the package is an python-based library (set of procedures), and it is run in a python interpreter. We recommend to use *ipython* as an interpreter, but the standard python should also work. Once you have started ipython, you need to import the module. To do this type: `import MagicTools`, if no error

message appeared, the importing was done correctly.

Analysis usually include the following phases: Reading the data from output files; Plotting the data; Numerical analysis/evaluation of the data; Export of the data.
Below we will discuss how these actions can be taken by MagicTools.

3.5.1 Reading the data

MagicTools can read data from several file types, used in MagiC: RDF and potential files ***.rdf, *.pot** and the MagiC core log file magic.out. This is done by procedures: `ReadRDF`, `ReadPot` and `ReadMagiC`, respectively.

Example:

```
import MagicTools
RDFs_ref=MagicTools.ReadRDF('MT1MT2.rdf')
Pots=MagicTools.ReadPot('01.MT1MT2.i010.pot')
RDFs_smpl=MagicTools.ReadMagiC('01.magic.out', iters=(1,2,10))
```

The first line imports the library, while the rest lines are showing how to call the reading procedures. The procedures put the data into specified variables: `RDFs_ref`, `Pots`, `RDFs_smpl`. The first two variables can be considered as a set of RDFs (or Potentials), having all non-bonded, pairwise and angle-bending distributions (potentials) provided in the file (and the occasional included files). The third variable, resulting from reading MagiC.core output is a list of RDFs, where every element of the list is a set of RDFs sampled while given iteration, as specified in parameter `iters`. The `ReadMagiC`, is not limited to reading just RDFs, but it can also extract potentials, corrected potentials, corrections applied at an iteration, reference RDFs. Check the procedure references for more details.

NB! Technically speaking, RDFs_ref and Pots are objects of class DFset (RDFs_smpl is a list of DFset objects), which contains basic properties and methods, allowing to deal with such set of function in somewhat simplified manner. For more details check the section referring to `DFset` class.

In addition to reading from the MagiC data files, procedures `LoadDFs` can be used to read intermediate DFs and lists of DFs, which were dumped on disk by `DumpDFs`.

3.5.2 Plotting and Inspecting the data

After the data are imported, they can be visualized via plotting. There are two possible ways one can plot datasets (RDFs, potentials, corrections, etc.) in MagicTools. The first one is to use general plotting function `PlotAllDFs`. This is especially handy when few DFs sets shall be plotted simultaneously, such as if one needs to compare RDFs sampled on different iterations.

Another way is to use `Plot` method of either `set of DFs` or of particular RDF/potential therein. This is convenient when one needs to plot one particular set of functions or even individual function.

Examples:

- Plot the set of RDFs, with one curve per plot:
  `MagicTools.PlotAllDFs(RDFs_ref)`

- Plot a list of sets of RDFs sampled at different iterations:
  `MagicTools.PlotAllDFs(RDFs_smpl)`
  in this case all the RDFs will be automatically grouped by the interaction they refers to.
- Plot together a list of DF-sets (e.g. sampled RDFs) and an additional DF-set (e.g. reference RDFs)
  `MagicTools.PlotAllDFs(RDFs_smpl+[RDFs_ref])`
- Alternative way to plot a set of RDFs (e.g. reference RDFs), with one curve per plot: `RDFs_ref.Plot()`

  Note the syntax: You state the actual name of the variable containing the set of DFs imported beforehand (see Reading data section above), and then use an embedded Plot() method of this variable.
- Plot a single function (RDF or potential) form the set:
  `RDFs_ref[0].Plot()`

  Note the syntax: As above, first you address the variable with the DFs-set (RDFs_ref), and then you specify the index[0] (counted from 0) of the function you would like to plot. Then you use an embedded Plot() method of this function.

In addition to plotting of already imported data, one can make a quick inspection of the MagiC core log-file using two procedures:

Procedure `Deviation` plots total deviation between reference and computed on different iterations distribution functions. Procedure `AnalyzeIMCOuput` plots reference and resulting DFs obtained in inverse procedure. They both require magic's output file as input.

Procedure `PlotAllDFs` takes a list of distribution functions (RDF, potentials) and plots them in relevant groups. This is the easiest way to plot and compare few sets of DFs, for example from different iterations of IMC.

3.5.3 Numerical analysis of the potentials

MagicTools has several procedures performing simple analysis of the effective potentials resulted from MagiC core:
`TotalPots` returns a set of total potentials,where the electrostatic interactions are included into the short-range intermolecular potentials.
`GetOptEpsilon` calculates the optimal value of dielectric permittivity which provides fastest decay of short-range intermolecular potentials at their tail.
`PotsEpsCorrection` creates a new set of potentials, where all non-bonded short range potentials are corrected to correspond to the new value of dielectric permittivity.
`PotsPressCorr` creates a new set of potentials, where all non-bonded short range potentials get added a decaying linear term, which suppose to improve total pressure in the target system.

3.5.4 Saving the data

Two operations are available: `DumpDFs`- dumps a list of DFs to a binary dump file, which can be read later by `LoadDFs`; `SaveDFsAsText` saves a distribution

function from a given list to a separate text-file;

3.5.5 Exporting potentials: GROMACS

Preparing a set of files for MD simulation with GROMACS is quite a challenge even for an experienced user. MagicTools provides few exporting procedures which hopefully make your life a bit easier. GROMACS requires provide five files: starting geometry (.gro, .pdb) and index file (.ndx), topology file (.top), simulation parameters (.mdp) and tabulated potentials (.xvg)

`xmol2gro` - converts `.xmol` structure file into GROMACS supported `.gro` format. Another workaround is to use VMD for opening `.xmol` and saving it in `.pdb` format. Once the starting structure is prepared, the index file can be generated by `make_ndx` tool, which is a part of GROMACS. Note that you shall create an individual group in the index file for every bead/atom type present in the system.

`GromacsTopology` - a simple tool to create a topology file, having the same interactions as in MagiC. Note that you shall manually state the actual number of molecules to be present in the system in the generated .top file.

`PotsExport2Gromacs` exports the list of potentials to a GROMACS's `.xvg` tabulated potential format, and also generates strings `energygrps` and `energygrp_table`, which shall be stated in .mdp file.

We highly recommend to check HowTo Tabulated potentials and respective part of the MagiC tutorial for getting some practical experience of the potential exporting procedure.

3.6 *MagicTools* procedures reference:

Reading: `ReadRDF`, `ReadPot`, `ReadMagiC`, `LoadDFs`
Plotting and inspecting: `PlotAllDFs`, `DFset.Plot`, `Deviation`, `AnalyzeIMCOuput`
Analyzing: `TotalPots`, `PotsEpsCorrection`, `GetOptEpsilon`, `PotsPressCorr`
Saving/Writing: `SaveDFsAsText`, `DumpDFs`, `SplitDFset`
Exporting2GROMACS: `PotsExport2Gromacs`, `GromacsTopology`, `xmol2gro`
Converting: `tpr2mmol`, `xmol2gro`, `gro2xmol`, `Convert2NewFile_pot`, `Convert2NewFile_rdf`

3.6.1 ReadRDF(ifile)

Read (import) reference distribution functions (RDFs) from a `.rdf` file **ifile*** (in MagiC 2.0 format) into a variable of the class `DFset` representing a single set of functions (which can be RDFs, potentials, potential corrections).

Example: `RDFs_ref=MagicTools.ReadRDF('MT1MT2.rdf')`

3.6.2 ReadPot(ifile, Ucut=1000)

Read (import) tabulated potentials from a `.pot` file **ifile*** into a variable of the class `DFset`. The optional parameter **Ucut** is a height of the hard repulsive core at r=0 in kJ/mol.

Examples:

```
Pots=MagicTools.ReadPot('01.MT1MT2.i010.pot')
Pots=MagicTools.ReadPot('01.MT1MT2.i010.pot', Ucut=5000.0)
```

3.6.3 ReadMagiC(ifile, iters=None, DFType='RDF', PairNamesList=None, mcmfile=None)

Parse the file **ifile*** produced by MagiC software and return a list of DFset (few sets of distribution functions: RDF/potential etc.): DFs[iteration][pair]

Optional parameters:

DFType - defines what shall be extracted from the file: 'RDF' - distribution functions sampled in MC calculations (one set for each iteration); 'RDFref' - reference RDFs(ADFs); 'Pot' - Potentials used for MC sampling in the iterations; 'PotNew' - resulting potentials generated in the iteration; 'PotCorr' - Potential correction applied in the iteration.

iters - tuple/list of iterations to extract from the file. If nothing mentioned all iterations will be extracted. Example: iters=(1,2,3) or iters=(1)

mcmfile - list of the mcm files (or a single file) used in the inverse MC calculation. If not provided, it will be generated automatically.

PairNamesList - List of pairs of atomic names to search in the output file. It consists of three sublists: first of them refer to non-bonded pair interactions, the second refers to bonded pair interactions, and the third one refers to bending angle (1-3) bond interactions. If no list provided, it will be generated automatically from the specified mcmfiles.

Examples:

1. Fully automatic: Read RDFs for all pairs and bonds and from all iterations. Autodetect mcmfiles.

```
RDFs=MagicTools.ReadMagiC('03.magic.out')
```

2. Reading potentials on iteration 1,2,3,

```
Pot=MagicTools.ReadMagiC('03.magic.out', iters=(1,2,3), DFType='Pot',
PairNamesList=[['N-N','N-P'], ['N-P','P-C1'], ['N-C1-P']])
```

3. Reading corrections to the potentials applied on the iteration 5, and specify the mcmfile

```
PotCorr=MagicTools.ReadMagiC('03.magic.out', iters=(5), DFType='PotCorr',
mcmfile='dmpc_NM.CG.mcm')
```

3.6.4 LoadDFs(filename)

Loads a set of DFs from a file previously dumped with `DumpDFs`.

filename* - name of the file to load (mandatory argument)

Example: `Pots=MagicTools.LoadDFs('pots.dmp')`

3.6.5 PlotAllDFs(listDFset, hardcopy=False, title='', linetype='', coinciding=False, nolegendintra=False, figsize=(20,14), dpi=80)

Plots a set (or list of such sets) of functions (RDFs, potentials, corrections). Functions can be plotted all together on a same plot, or one per plot, or grouped by the type of interaction they are representing.

listDFset* - list of sets of distribution functions, e.g. it is a list which keeps a few DFset objects inside (mandatory argument). Can be also a single set

of functions

hardcopy - If the plots should be saved as *.eps files (optional argument). Default value - False, no eps copies are made.

title - Prefix used in a title of each plot (optional argument).

linetype - String defining a type and color of lines according to matplotlib syntax (optional argument). See more about syntax here: matplotlib tutorial

coinciding - Plot only coinciding functions (i.e. related to the same pair of bead types, or related to the same bond within the same molecule). Useful when one need to compare similar functions obtained from different (but close to each other) systems.

nolegendintra - If a legend on intramolecular plots should be omitted (optional argument). Default:False - show the legend everywhere. This option might be useful, when legend overlaps with curves on plot, usually it can happen when plotting intramolecular DF/potentials.

figsize - Size of the plot in inches (x,y) (optional argument). Default size is (20,14).s

dpi - Resolution of the plot in dpi (optional argument). Default value: 80 dpi.

Examples:

```
RDFs=MagicTools.ReadMagiC('03.magic.out')
RDFref=MagicTools.ReadMagiC('03.magic.out',DFType='RDFref')
```

Import RDFs obtained on every iteration of IMC to a list of `RDFs` (list of DFset objects) and also import reference RDFs

`MagicTools.PlotAllDFs(RDFs)` - Plot RDFs from the list at the same plot grouped by origin (pair of beads involved in function).

```
MagicTools.PlotAllDFs([RDFref]+[RDFs[0]],hardcopy=True,
 title='RDF convergence in IMC',linetype='.')
```

Plot RDFs calculated in the first iteration of IMC at the same plot with reference RDFs, save copies to eps-files. Plots will be drawn with dots instead of lines, and a title will be added to each plot.

`MagicTools.PlotAllDFs(RDFref)` - Plot every function for RDFref on a separate figure.

3.6.6 DFset.Plot(AtOnce=False, hardcopy=False, title='', linetype='', nolegendintra=False, figsize=(20,14), dpi=80):

Plots all functions stated in the set *DFset* either on one plot or on few individual plots.

AtOnce - if true, all functions are plotted on the same plot (separated by type: i.g. 3 plots are produced: NB, B and A), otherwise each function is plotted separately

hardcopy - If the plots should be saved as *.eps files (optional argument). Default value - False, no eps copies are made.

title - Prefix used in a title of each plot (optional argument).

linetype - String defining a type and color of lines according to matplotlib syntax (optional argument). See more about syntax here: matplotlib tutorial

nolegendintra - If a legend on intramolecular plots should be omitted (optional argument). Default:False - show the legend everywhere. This option might be useful, when legend overlaps with curves on plot, usually it can happen when plotting intramolecular DF/potentials.

figsize - Size of the plot in inches (x,y) (optional argument). Default size is (20,14).

dpi - Resolution of the plot in dpi (optional argument). Default value: 80 dpi.

Examples:
`RDFref.Plot()` Plot all functions imported in the variable `RDFref` (i.e. reference distribution functions), each function on an individual plot.
`RDFref.Plot(AtOnce=True)` Plot all functions on the same plot. One plot per function type will be produced: Non Bonded, Bonded and Angle bending.

3.6.7 Deviation(filename,hardcopy=False,returnarrays=False, testpoints=False)

Analyze the output file **filename*** (or list of files) produced by the MagiC core and plot deviation between the set of reference distribution functions and sampled distribution function obtained on every iteration of the inverse procedure. Two deviations are calculated:
$\Delta S \sim [\sum_{r_j=0}^{r_j=r_{max}} (S_{iter}(r_j) - S_{ref}(r_j))^2]^{0.5}$ and
$\Delta RDF \sim [\sum_{r_j=0}^{r_j=r_{max}} (g_{iter}(r_j) - g_{ref}(r_j))^2]^{0.5}$
If an intermediate convergence test has been performed during inverse procedure, results of the test are also plotted.

filename* - name of the magic output file or list of such names (mandatory argument).

hardcopy - if the plot should be saved to a .eps file (optional argument). Default - no.

testpoints - if the points sampled in intermediate convergence tests shall be also plotted. Default - no.

returnarrays - if true, the procedure returns numpy arrays with iteration number and deviation values.

Examples:
```
MagicTools.Deviation('01.magic.out')
MagicTools.Deviation(['01.magic.out','02.magic.out'],hardcopy=True)
```

3.6.8 AnalyzeIMCOuput(filename, DFType='RDF', iters=None, hardcopy=True, mcmfile=None, PairNamesList=None)

Quick tool to parse the output file produced by MagiC core and plot the sampled DFs of interest (and the reference DFs). The parameters have exactly the same meaning as in ReadMagiC
Example:
```
MagicTools.AnalyzeIMCOuput('01.magic.out')
```

3.6.9 TotalPots(pots, eps, mcmfile=None)

Creates a set of total potentials by adding electrostatic interactions to a short-range potential.

$$U_{tot} = U_{sr} + \frac{q_i * q_j}{4\pi\epsilon\epsilon_0 r_{ij}} \tag{2}$$

Electrostatic part is only applied to the intermolecular potentials, while bond potentials (both pairwise and angular) will be kept the same.

pots* - list of potentials (mandatory argument)

eps* - dielectric permittivity ϵ of implicit solvent used in inverse Monte-Carlo simulation (mandatory argument).

mcmfile - molecular description file (or list of files) providing charges for bead/CG-atom types. Required if the potential was read from .pot file rather than from MagiC core log file.

Example:
`Pots=MagicTools.ReadPot('03.magic.i010.pot')` - import potentials.
`TotalPots=MagicTools.TotalPots(Pots, eps=70.0, mcmfile='dmpc_NM.CG.mcm'))`
Create total potentials and store it in the variable `TotalPots`.

3.6.10 GetOptEpsilon(pots, eps_old, r1, eps_min=0, eps_max=0, npoints=100, mcmfile=None)

Calculates optimal value of the dielectric permittivity which provides fastest decay of short-range intermolecular potential tales according to the procedure described in: A.A.Mirzoev and A.P.Lyubatsev, Phys.Chem.Chem.Phys., 13, 5722-5727 (2011) DOI: 10.1039/C0CP02397C.

Briefly, the procedure to obtain values of the dielectric permittivity providing fastest decay of short-range potentials set with distance consists in the following. First, we introduce a numerical criteria of a short range potential deviation from zero at large distances:

$$W(U_{sr}^{ij}(r)) = \int_{r_1}^{r_2} \left| r^2 (U_{sr}^{ij}(r)) \right| dr \tag{3}$$

where r^2 factor implies a higher weight of larger distances, r_1 and r_2 are the lower and upper boundaries of the range of distances defining the tail (the r_2 value is taken as the cut-off of RDFs and tabulated effective potentials). The absolute value in the equation is used in order to deal with possible oscillations of the short range part of the potential. From eq. 2, one can write for the short-range part of the potential:

$$W(U_{sr}^{ij}(r)) = \int_{r_1}^{r_2} \left| r^2 (U_{tot}^{ij}(r) - \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon r}) \right| dr \tag{4}$$

Assume we define the long-range Coulombic potential using another value of permittivity $\varepsilon^\star$. This, according to 2, introduces a new short-range potential as:

$$U_{sr}^{\star\, ij} = U_{sr}^{ij}(r) + \frac{q_i q_j}{4\pi\varepsilon_0 r}\left(\frac{1}{\varepsilon} - \frac{1}{\varepsilon^\star}\right) \tag{5}$$

Now we shell find the optimal $\varepsilon^\star$, which produces the fastest decay of all three short range potentials according to criteria defined by eq. 3. We minimize the sum:

$$W(system) = \sum_{i,j} W(U_{sr}^{\star\, ij}(r)) = \sum_{i,j} [U_{sr}^{ij}(r) + \frac{q_i q_j}{4\pi\varepsilon_0 r}(\frac{1}{\varepsilon} - \frac{1}{\varepsilon^\star})] \qquad (6)$$

by varying $\varepsilon^\star$. The optimal value of $\varepsilon^\star$ can be considered as effective dielectric permittivity corresponding to the given thermodynamic conditions (temperature, concentration).

pots* - set of potentials to analyze (mandatory argument) NB! The dielectric permittivity value calculation only takes intermolecular potentials into account skipping bonding potentials.

eps_old* - dielectric permittivity used in inverse MC calculation (mandatory argument)

r1* - distance where tail range begins, Å(mandatory argument)

eps_min, eps_max - range of values for the search of ϵ_{opt} (optional argument). By default eps_min=0, eps_max=2*eps_old

npoints - number of points in a mesh to be used for the search, e.g. accuracy of the search is equal to $\frac{\epsilon_{max}-\epsilon_{min}}{npoints}$

mcmfile - molecular description file (or list of files) providing charges for bead/CG-atom types. Required if the potential was read from .pot file rather than from MagiC core log file.

Example:
`eps_opt=MagicTools.GetOptEpsilon(Pots, 70.0, 15, eps_min=50, eps_max=100, mcmfile='dmpc_NM.CG.mcm')` - get optimal epsilon value.

3.6.11 PotsEpsCorrection(pots, eps_old, eps_new, mcmfile=None)

Creates a new set of potentials, where intermolecular potentials are adjusted to a changed value of the dielectric permittivity according to eq.5. Intramolecular (bond) potentials are kept untouched.

pots* - set of potentials to analyze (mandatory argument) NB! The correction only affects intermolecular potentials.

eps_old* - dielectric permittivity used in inverse MC calculation (mandatory argument)

eps_new* - new value of dielectric permittivity.

mcmfile - molecular description file (or list of files) providing charges for bead/CG-atom types. Required if the potential was read from .pot file rather than from MagiC core log file.

Example: `newpots=MagicTools.PotsEpsCorrection(Pots,eps_old=70,eps_new=100)`

3.6.12 PotsPressCorr(pots,U_corr0)

Creates a new set of short-range potentials by adding a decaying linear term to each intermolecular potential in the set. Such a correction suppose to improve reproduction of a correct pressure in the large scale CG simulation. Intramolecular potentials are kept untouched. Correction term is linear and has value of U_{corr0} at point r=0, and value of 0 at $r = r_{max}$, e.g. $U_{corr}(r) = U_{corr0} \cdot (1 - \frac{r}{r_{max}})$

pots* - set of potentials to analyze (mandatory argument) NB! The correction only affects intermolecular potentials.

U_corr0* - Magnitude of the correction, kJ/mol (mandatory argument)

Example: `newpots=MagicTools.PotsPressCorr(Pots,0.5)`

3.6.13 SaveDFsAsText(DFs)

Save every function from a given set of functions (or a list of such sets) into a separate text-file. Each function is saved in a tabulated format: First column - distances in Å, second column - values. The text file has the same name as the according function. The files are ready to be plotted by `gnuplot`, e.g.
`gnuplot> plot './NB.RDF.NB.N-N.i1.dat' w lines`

DFs* - set (list of sets) of the functions to save (mandatory argument)

Example: `MagicTools.SaveDFsAsText(Pots)`

3.6.14 DumpDFs(DFs,filename)

Dumps a given set of DFs into a file. The file can be read using LoadDFs.

DFs* - set of distribution functions to dump (mandatory argument)

filename* - name of the dump-file (mandatory argument)

Example: `MagicTools.DumpDFs(Pots,'pots.dmp')`

3.6.15 SplitDFset(ifile, ofile, Split=True)

Splits the set of RDFs or potentials given in the file ifile into a main header file and an additional set of included files.

DFs* - set of distribution functions to dump (mandatory argument)

filename* - name of the dump-file (mandatory argument)

Split - logical list, stating which DFs within the file shall be moved to included file, while other will remain in the main header file.

Examples:

```
MagicTools.SplitDFset('03.magic.i010.pot','magic.pot')
MagicTools.SplitDFset('03.magic.i010.pot','magic.pot', Split=[True,False])
MagicTools.SplitDFset('03.magic.i010.pot','magic.pot',
 Split=[i>10 for i in xrange(0,20)])
```

3.6.16 GromacsTopology(mcmfile, topfile='topol.top')

Creates a GROMACS topology file *.top from a given of mcm-file or a list of mcm-files. Number of molecules in the resulting system should be stated manually once the top-file is created.

mcmfile* - Name of the mcm-file or a list of such files (mandatory argument).

topfile - Name of the output GROMACS topology file. Default: topol.top

Examples:
`MagicTools.GromacsTopology('dmpc_NM.CG.mcm')` - system of single molecular type
`MagicTools.GromacsTopology(['dmpc.mcm','dopc.mcm'],'dmpc_dopc.top')` - system having two different molecular types

3.6.17 PotsExport2Gromacs

PotsExport2Gromacs(pots, npoints=2500, Umax=6000, Rmaxtable=2.5, PHImaxtable=180, filename='', noplot=False, hardcopy=True, figsize=(14, 7.5), dpi=120, sigma=0.5, zeroforce=True, interpol=True, ofilename='')

This procedure exports a set of potentials into GROMACS's .xvg format. Such an export is a rather advanced and multistage process, here a detailed description of the procedure is given.

The first thing to take into account is that Gromacs requires typically a higher resolution of the grid for tabulated potentials than typical resolution of the RDF inversion which is used in MagiC. Furthermore, the potential and force should be smooth functions of the distance, and do not have discontinuities at the end of the intervals where they are determined. Thus the original range where the potentials and RDFs were determined during the inversion procedure, should be extended in a smooth fashion to nearby range of distances.

Thus in general the question is the following: we start with a tabulated potential which is defined on a r-grid $[r_{min} : r_{max}]$ with a given density (figure 4-A, red circles); and as the result we need to obtain a tabulated potential, which is defined on a r-grid $[0 : r_{vdw} + r_{table-extension}]$ with larger density (figure 4-D, cyan line).

In order to do that, we need to introduce left-side and right-side extensions of the potential which should be smoothly connected to the original potential (figure 4-B, blue circles). The left side extension should represent strongly repulsive core so it is approximated by

$$U^{left}(r) = ar^2 + br + U_{max} \tag{7}$$

and coefficients a and b are chosen to provide continuity of $\frac{d}{dr}U(r)$ at $r = r_{min}$. For the right side extension we should take into account the origin of the potential: Short-range intermolecular potentials should decay to zero when $r > r_{max}$:

$$U^{right}_{short\,range}(r) = U(r_{max}) \cdot \exp[-10\frac{(r - r_{max})}{r_{vdw+table-extension} - r_{max}}] \tag{8}$$

Here 10 is just some pre-defined coefficient, $r_{vdw+table-extension}$ is a range of the table required by GROMACS. Angle bending potentials should also decay to zero at $\phi = 180°$:

$$U^{right}_{angle}(\phi) = U(\phi_{max}) \cdot \exp[-100\frac{(\phi - \phi_{max})}{180° - \phi_{max}}] \tag{9}$$

In contrast, pairwise bond potentials should have an attractive wall at the right side, which is approximated by harmonic wall in the same way as the repulsive wall at the left side:

$$U^{right}_{pair\,bond}(r) = ar^2 + br + U_{max} \tag{10}$$

coefficients a and b are chosen to provide continuity of $\frac{d}{dr}U(r)$ at $r = r_{max}$.

Once the extension has been made, we can interpolate all the points to a denser grid (figure 4-C). Number of nodes in the grid is defined by `npoints`. If `interpol=False`, a grid of original density is used. The interpolation is made by Gaussian smoothing, e.g. every point of the resulting potential is obtained as

$$U^{new}(r) = \frac{1}{Z(r)} \sum_{r_i=r_{min}}^{r_{max}} U^{orig}(r_i) \cdot \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{11}$$

$$Z(r) = \sum_{r_i=r_{min}}^{r_{max}} \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{12}$$

where sigma defines how broad is the averaging. By default $\sigma = 0.5\Delta r_i$

Once the interpolation is done, each resulting potential and force based on it are written to a xvg-file. The xvg-file is named by the name and type of the corresponding original potential. Forces can be suppressed by setting `noforce=True`, then zeros will be written to the file. In such case GROMACS should automatically calculate forces from a given potential.

In order to control the results, each original potential, the extrapolated part and the interpolated potential are plotted together. The plotting is controlled by parameters `noplot, hardcopy, figsize, dpi`, which are explained below.

pots* - set of potentials to export into GROMACS .xvg format (mandatory argument)

ofilename - Prefix used for naming of the output .xvg files (optional argument).

zeroforce - do not write forces into .xvg-file, but write zeros instead (optional argument). In such case GROMACS should automatically calculate forces from potentials. Default: False - forces are to be written. NB: Even if `zeroforce=True` force values are plotted.

npoints - number of points in the resulting table in .xvg file (optional argument). Default value - 2500 points.

Umax - height of potential wall at r=0 in case of non-bonding potential and at $r = r_{min}$, $r = r_{max}$ in case of bonding potential (optional argument). Default value 6000 kJ/mol

Rmaxtable - cutoff range of the resulting potentials (both intermolecular and pair bonds) in nanometers (optional argument). Default value is 2.5 nm. Note that GROMACS requires tabulated potentials to be defined up to r=rvdw+table-extension (in terms of GROMACS parameters).

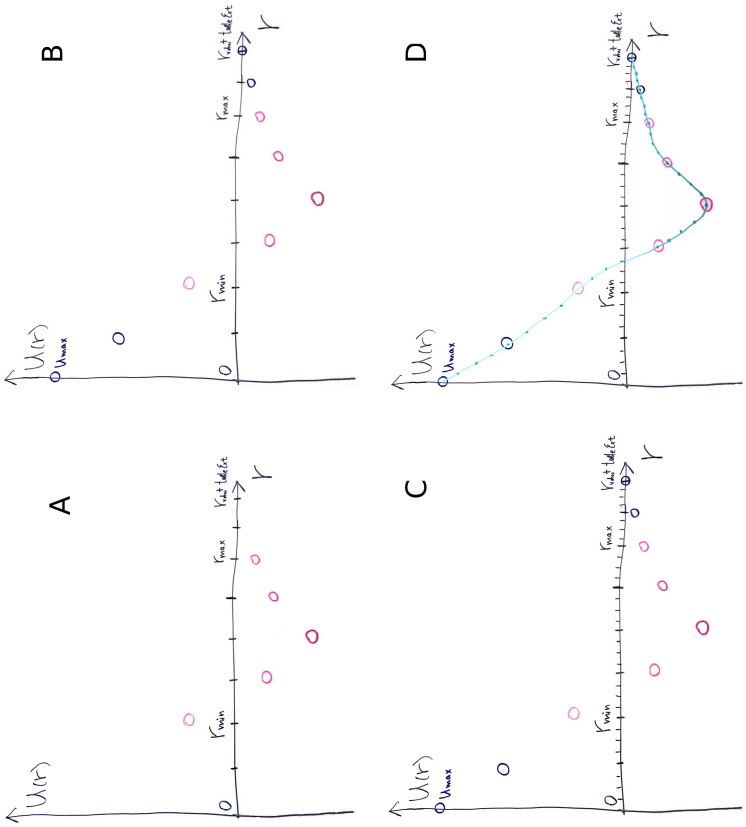

Figure 4: Original potential - red circles

sigma - Gaussian smoothing parameter measured in resolution of the original potential (optional parameter). Default value 0.5

interpol - if the potentials should be interpolated, otherwise original resolution of the table will be kept and `npoints` value will be ignored (optional argument). Default - True, potentials are interpolated.

noplot - if interpolated potentials potential/forces should be plotted (optional argument). Default: False - the graphs are plotted.

hardcopy - if the plots should be saved as *.eps files (optional argument). Default value - True.

figsize - size of the plot in inches (x,y) (optional argument). Default size is (14,8).

dpi - resolution of the plot in dpi (optional argument). Default value: 120 dpi.

Examples:

`MagicTools.PotsExport2Gromacs(Pots)` - simplest call with default parameters.

`MagicTools.PotsExport2Gromacs(Pots[0:10], sigma=0.1, zeroforce=True, interpol=False )` - export only first 10 potentials from the set; do not use dense net for interpolation; use averaging with sigma=0.1 (very narrow gaussian); do not export

forces, write zeros instead.

3.6.18 xmol2gro

Converts `*.xmol` file **ifilename*** to a `*.gro` file **ofilename***.

molnames* - list of molecular types that are present in the system

nmols* - list stating how many molecules of the respective type are present in the system

natimol* - list stating how many atoms each molecular type has

nconf - how many configurations to write.

Example:
```
MagicTools.xmol2gro(input.xmol, output.gro, molnames=['DMPC','WAT'],
 nmols=[98,2700], natimol=[118,3], nconf=1)
```

3.6.19 gro2xmol(ifilename, ofilename='output.xmol')

Convert `*.gro` file **ifilename*** to a `*.xmol` file **ofilename**. **Example:**
```
MagicTools.gro2xmol(ifilename, ofilename='output.xmol')
```

3.6.20 tpr2mmol(tprfile)

Convert GROMACS binary run input file ***.tpr** to a set of ***.mmol** files. Useful for preparing input data for the bead mapping with CGTraj or for the RDFs calculation.
Example:
```
MagicTools.tpr2mmol('mdrun.tpr')
```

3.6.21 Convert2NewFile_pot(potfile,mcmfile)

Convert an old *.pot file from MagiC-1.0 format to the MagiC-2.0 file format. The generated potential-file will have same name as an old one, with additional suffix ".v2."

potfile* - filename of the original "old" ver 1.0 file

mcmfile* - file or list of mcm-files (or mmol-files), in the same order as they were introduced in the respective IMC calculation where the potential was obtained.

Example:
```
MagicTools.Convert2NewFile_pot('01.dmpc16-100a.i001.pot','dmpc_NM.CG.mcm')
```

3.6.22 Convert2NewFile_rdf(rdffile, mcmfile)

Convert an old *.rdf file from MagiC-1.0 format to the MagiC-2.0 file format. The generated rdf-file will have same name as an old one, with additional suffix ".v2."

rdffile* - filename of the original "old" ver 1.0 file

mcmfile* - file or list of mcm-files (or mmol-files), in the same order as they were introduced for rdf-calcualtion

Example:
```
MagicTools.Convert2NewFile_rdf('dmpc16-100aa-rdf-mdyn.rdf','dmpc_NM.CG.mcm')
```

3.6.23 DFset - set of Distribution Functions

DF set is an object class representing a set of Distribution Functions (RDF,Potential,Potential correction, etc.).

Properties:

Name - Name of the set (typically name of the file the set was read from)

Kind - Kind of functions collected in the set: RDF or POTENTIAL

NTypes - Number of different atom/bead types used in the set

Types - Names of the atom/bead types

Min, Max - Range of distance values for non-bonded interaction functions

Npoints - Number of points in non-bonded interaction functions

DFs - List of functions (all functions in the set)

DFs_NB - List of non-bonded interaction functions

DFs_B - List of pairwise bond interaction functions

DFs_A - List of angle-bending bond interaction functions

Methods:

DFset() - Construct the object from provided rdf/pot file (MagiC 2.0 format) or from the provided parameters (old fashion)

Write() - Write the set of functions to the file (.rdf or pot).

Plot() - Plot the set of functions

Examples:
`import DFset` - import the class
`RDFref=MagicTools.ReadRDF('dmpc.400ns.v2.rdf')` - read a set of RDFs, which is and object of the class DFset.
`RDFref2=DFset.DFset('dmpc.400ns.v2.rdf')` - other way to read the set, giving exactly the same result
`RDFref2.Plot()` - Plot the functions.
`RDFref2.Write('RDFref2.rdf')` - Write the set to the file.
`RDFref2.Name` - Access the specific property of the set (Name)
`RDFref2.DFs[0]` - Access the first function of the set (they are indexed from 0)
`RDFref2.DFs_B[0]` - Access the first pairwise bond related function of the set (they are indexed from 0)

3.6.24 DF - Distribution Function

DF is a object-oriented class, which is supposed to reproduce basic features of a single distribution function (e.g. RDF, bond length distribution, angle distribution, intermolecular potential, angle-bending potential, correction to potential, etc.).

The class contains properties and methods, which are common for every function, however, some methods are redefined when necessary to keep function specificity:

Properties:

Name - Name of the function

Kind - Kind of the function: 'RDF' or 'POTENTIAL'

Type - Type of the function: It can be 'NB', 'B' or 'A', for Non-bonded, pairwise bond and angle bond, respectively.

Npoints - Number of points in a table defining the function.

Min,Max - Range of distances(Å)/angles(deg) where the function is defined

Resol - Resolution of the table

g[:, 0:2] - The table (numpy array) defining the function. g[:,0]-keeps argument (r,angle), g[:,1]-keeps function's value

AtomTypes - (NB only) Names of the atom/bead types involved in the interaction represented by the function

BondNumber - (B/A only) Number of the bond represented by the function.

MolTypeName - (B/A only) Name of the Molecular type the bond function refers to.

AtomGroups - (B/A only) List of atom pairs/triplets involved in the bond

Methods:

Plot() - Plot the function, using matplotlib library

Save() - Write the function in a tabulated format to a text file.

Smooth() - Smooth values of g[:,1] using Savitzky-Golay 5 points smoothing procedure

Trim(tolerance) - Cut function's (g[:,1]) left and right tails which have values smaller then tolerance

Examples:
`import DF` - import the class
`RDFref=MagicTools.ReadRDF('dmpc.400ns.v2.rdf')` - read a set of RDFs, which is and object of the class DFset, containing a number of DF-objects.
`rdfNB=RDFref.DFs[0]` - Access the first function of the set. It is an object of class DF representing a non-bonded RDF.
`rdfNB.g` - Access the table of the function
`rdfNB.Plot()` - Plot the function
`rdfNB.AtomTypes` - See what bead/atom types are involved in the function.
`rdfB=RDFref.DFs_B[0]` - Access the first pairwise bond related function of the set (they are indexed from 0)
`rdfB.MolTypeName` - See the name of the molecular type the bond function belongs to.

4 File formats

4.1 .xmol

This is plain text single configuration or trajectory file format, which can be produced by many molecular modeling packages, including MDynaMix, MagiC, or converted from *.gro by gro2xmol.py script subsubsection 3.6.19. It consists of a number of consequent frames, with each frame having the following structure:

line 1: Number of atoms in the frame (N)

line 2: A commentary line

line 3: Name(atom1) X(atom1) Y(atom1) Z(atom1)

line 4: Name(atom2) X(atom2) Y(atom2) Z(atom2)

lines 5,6...,N+2: Names and coordinates of atoms 3 - N.

In case of trajectory, configuration files from each time frame are written consequently one after another.

There is no common requirements for the commentary (second) line. For CGtraj module of MagiC it is assumed that the second line of each configuration follows this format (accepted in MDynaMix):

`(char) <time> (char-s) BOX: <box_x> <box_y> <box_z>`

where `(char)` is any character word, `<time>` is time in fs, `<box_x> <box_y> <box_z>` (following after keyword `BOX`). The length unit is Ångströms and time unit is femtoseconds. The time step information is not needed for CGTraj but the box size information is essential. If no box size information is present in the trajectory, the box size information can be supplied in the input file, but the later has a sence only in constant-volume simulations.

In xmol-files produced by MagiC the second line may have no time-stamp and box sizes.

4.2 .mmol

MMOL is a molecular topology file format, which is inherited from MDynaMix MD software. It consists of two parts: first part describes atomic composition, geometry, charges, masses and non-bonded interactions; the second part defines bonds, angles and torsions in the molecule. MMOL-topolygy files are required by cgtraj (subsection 3.2) for converting high-resolution trajectory to a coarse grained one and for calculation of the reference distribution functions rdf (subsection 3.3). In both cases only information from the first section of .mmol file (information about atomic composition) is used, and the bonding part may be omitted.

The first part of the file, which is part of interest has the following structure: The first non-commentary line of a .mmol file is the number of atoms in the molecule. After it the corresponding number of lines follows, one line per atom. Each line contains 8 compulsory parameters. They are: 1) atom name in the program; 2),3) and 4) are the initial X,Y,Z coordinates of the atom in the molecular coordinate system, 5) mass in atom units, 6) charge, 7) Lennard-Jones parameter σ in , 8) Lennard-Jones parameter ε in kJ/M. Two optional columns may present. Note also, that for the correct work of CGTraj utility, the only important information is the number of atoms in the molecule and masses of atoms.

4.2.1 MMOL Example: H2O.mmol

```
#================================================I
#       Molecular Dynamics Data Base             I
#       Configuration and interaction potential  I
#================================================I
```

```
#           SPC H2O model                          I
#================================================I
#       Number of sites
 3
#       X        Y         Z         M          Q         sigma     epsilon
#               (A)                                                (kJ/M)
O       0.       0.    -0.064609  15.9994      -0.82       3.1656    0.6502
H1      0.    -0.81649  0.51275    1.008        0.41       0.        0.
H2      0.     0.81649  0.51275    1.008        0.41       0.        0.
# We care only up to this line! The rest of this file can be skipped.
#    Num. of strings for the reference
4
       SPC water model
       Parameters from:
       K TOUKAN AND A.RAHMAN,
       PHYS. REV. B Vol. 31(2) 2643 (1985)
#   Num. of bonds
 3
#ID(typ)    N1   N2         Reqv      Force         D        RHO (A**-1)
 1           1    2          1.       2811.      420.      2.566
 1           1    3          1.       2811.      420.      2.566
 0           2    3        1.633       687.       0.        0.
#   Num. of angles
 0
#   Num of dihedrals
 0
#   Additional options
#   flexible SPC water
fSPC
```

4.3 .mcm

Coarse-grain topology file, which is similar to CG.mmol, but includes information about bead/CG-atom types and intramolecular bonds. In general it consists of three parts: the first part describes atoms involved in the molecule, the second part contains list of covalent-like bonds and the third part lists angle-bending bonds. An individual .mcm file should be provided for each molecular type present in the system. These files are automatically generated by rdf.py utility during computation of th e reference distribution functions.

Format of .mcm file:

NB! Lines beginning with '#' or '!' are commentaries, they are dropped while parsing the file.

First the atom description block is specified:

1 line: Number of bead/atoms in the molecule (Natoms)

Natoms lines: Atom records. One record per line.

Each record contains 8 parameters. Atom name; X,Y,Z coordinates (Å) of the atom in a local coordinate system, mass (au.); charge (el.); index of the atom type; name of bead/atom type. The short-range non-bonded interaction between a pair of beads/atoms will be defined by the bead/atom types defined in this file. Atoms of the same atom type interact by the same non-bonded potential.

Then the pairwise bond block is specified:

1 line: The total number of pairwise bond types present in the molecule (Nbonds).
Then for every individual bond type (of Nbonds) one need to specify:
1 line: Number of atom pairs which are involved in the bond of this type (NPairs);
NPairs lines: List of such atom pairs, one pair per line.
Then the angle bending bond block is specified:
1 line: Total number of angle bending bond types present in the molecule. For every individual bond type (NAngles) one need to specify:
1 line: Number of atom triplets which are involved in the bond of this type (NAngles);
NAngles lines: List of such atom triplets, one triplet per line.

NB! In the first version of MagiC the triplet used to had unusual order: central atom stands last in the triple, e.g. triplet 1 3 2, defines angle between 1-2 and 2-3. Now it is changed to the regular 1-2-3 order, and to avoid misunderstanding `rdf.py` utility automatically writes `Order=1-2-3` after the number of total angle bending bonds.

4.3.1 MCM Example: DMPC.CG.mcm

The mcm-file listed below defines 10-beads model of DMPC-lipid, as shown on figure 5.

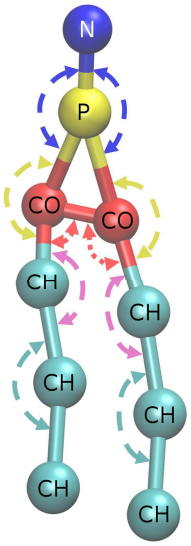

Figure 5: Example: 10-beads CG model of DMPC-lipid. Beads and bonds of same color have same type; solid lines denote covalent bonds; dashed arrows denote angle bending bonds.

```
#Number of atoms
10
# Name    X      Y       Z       Mass    Q  NumofType NameofType
N     -6.1804 -17.2679 17.2781 73.139 0.76 1 N
```

```
P    -9.6995 -17.2121 14.915 123.0256 -0.89 2 P
C2   -18.1023 -13.2828 10.2371 56.108 -0.0 3 CH
C3   -21.9375 -11.5562 7.3382 56.108 -0.0 3 CH
C4   -24.5552 -9.7683 3.2938 57.116 -0.0 3 CH
C6   -15.3122 -17.1232 8.1441 56.108 -0.0 3 CH
C7   -18.7644 -15.0198 5.1392 56.108 -0.0 3 CH
C8   -21.6201 -13.2988 1.0155 57.116 -0.0 3 CH
C1   -14.7182 -15.6667 12.7078 72.0638 -0.09 4 CO
C5   -13.3132 -18.7418 11.2877 71.0558 0.22 4 CO
#
# Here we define covalent like bonds
# Total number of covalent bond types: 5
5
# Covalent bond N-P
# One atom pair belongs to covalent bond type 1
1
# Define pair of atoms by their numbers in the list above: 1 (N) 2 (P),
# the third number should always be 1 for covalent bond.
1  2
# Covalent bond P-CO
# Two atom pairs belong to covalent bond type 2
2
# Define 2 pairs of atoms by their numbers in the list above: 2 (P) and 9,10 (C1,C5)
2  9
2  10
# Covalent bond CH-CH
# Four atom pairs belong to covalent bond type 3
4
# Define 4 pairs of atoms by their numbers in the list above: 3-4, 4-5, 6-7, 7-8
3  4
4  5
6  7
7  8
# Covalent bond type 4
2
9  3
10  6
# Covalent bond type 5
1
9  10
#
# Here we define angle bending bonds
# Total number of angle bending bonds:5
5
# Two atom triplets belong to angle bond type 1:
# OBS! Triplet has unusual order: central atom stands last in the triplet, e.g.
# triplet 1 9 2 means angle 1-2-9, with 1,9 on sides, and 2 on corner.
2
1  2  9
1  2  10
# Two atom triplets belong to angle bond type 2:
2
2  9   3
2  10  6
```

```
# angle bond type 3:
2
9  3  4
10  6  7
# angle bond type 4:
2
3  4  5
6  7  8
# angle bond type 3:
2
3  9  10
6  10  9
```

4.4 .rdf and .pot file formats

File formats for .rdf and .pot files are valid fo MagiC v. 2.0 and higher. The format of files for .rdf and for potentials is similar and we give here a common description of it refering as "RDF/potential" file format.

The RDF/potential file consists of a header section, marked by tags `&General` and `&EndGeneral` describing general properties of the RDFs/Potentials set for a specific system, and independent RDF/potential records, marked `&Potential` ... `&EndPotential` or `&RDF` ... `&EndRDF` respectively, which specify RDF/potentials for each individual interaction. Each individual RDF/Potential can be included to the RDF/potential file from a separate file by `include` statement.

Non-Bonded (NB) RDF/Potentials are identified by atom types involved in the RDF/Potential. Pairwise (B) and Angle-bending (A) bonds are identified by the molecular type they belongs to and the relative bond number in the molecular type. Each Potential can be protected from correction (Fixed) by specifying the flag Fixed=True in the corresponding section of the potential file.

4.4.1 Header

`&General - &EndGeneral`

The header defines common properties of the RDFs/potentials provided in the file. Since individual RDF/potential records can be included from external files, the header section provides important information which helps to provide consistency between all records.

NTypes - Number of bead/atom types present in the system.

N_NB, N_B, N_A - Number of Non-Bonded, pairwise Bonded, and Angle-bending records, respectively

NPoints - Number of points in each NB-record.

Min, Max - Range of distance (in Å) where Non-Bonded RDF/Potentials are defined

4.4.2 RDF/potential record:

`&RDF` ... `&EndRDF`
or
`&Potential` ... `&EndPotential`

Each individual record specifies one RDF/potential, providing information the CG atom types, range, number of points and the data-table with actual values. A record can be also included from a separate file. Every record consists of specifications, data table (`&Table...&EndTable`) and optional include section (`&IncludePotential...&EndIncludePotential`).

Name - Name of the RDF/potential record. Is used as a comment line

Type - Type of the record. Can be NB, B, A, i.e. non-bonded, pairwise bond, angle-bending bond, respectively

Min, Max - Range (in Å) where the record is defined

NPoints - Number of points in the record. Note, that for the core region of RDF, which has zero values and often omitted, the Min value is typically not zero

AtomTypes - For NB-record, specifies the pair of bead/atom types which are involved in the interaction.

MolType - For B- or A-record, specifies molecular type the bond belongs to.

BondNumber - For B- or A-record, specifies a relative number of bond to which this record belongs. Note that bonds are indexed locally, with respect to the molecular type, the same way as in corresponding mcm-file for the given molecular type

NPairs or NTriplets - Number of atom pairs (triplets) involved in the bond (B- or A-bond)

Pairs or Triplets - List of atom pairs (triplets) involved in the bond. Pairs/Triplets are comma separated, and atom numbers are separated by dash symbol (-) withing each pair/triplet. For triplets, atom numbers are specified in a direct way, so the central atom of the triplet is a central atom of the angle. Atoms are numbered locally, with respect to the molecular type, same way as in the corresponding mcm-file for the given molecular type).

&Fixed - Potential file only. If stated, the potential will be excluded from the inverse procedure (e.g. it will be fixed in potential update/refinement)

&InitZero - RDF-file only. If the corresponding potential is not provided, initiate it with zero. Default for NB-potentials.

&InitPMF - RDF-file only. If the corresponding potential is not provided, initiate it with Potential of Mean Force. Default for bond-potentials (both A- and B-).

&Table...&EndTable - Actual table defining the RDF/potential. The first column specifies distance (Å) or angle (deg), the second column specifies value, which is unitless for RDF and kJ/mol for potential. The table shall be uniformly spaced, and the same grid resolution should be used in all RDFs and potentials of the same kind (NB, B or A) for the whole system

&IncludePotential=IncludeFileName or

&IncludeRDF=IncludeFileName - Instead of providing data table in the record, one can import it from an external file ¡filename¿. This feature allows to easily incorporate potentials previously obtained for other systems into the current one.

4.4.3 Included Potential/RDF

`&IncludedPotential` ... `&EndIncludedPotential`
`&IncludedRDF` ... `&EndIncludedRDF` The included record has nearly identical format as the parental RDF/potential section. The record specification values shall be also in agreement with the corresponding values of the parental section, to provide consistency of the whole set of RDF/potentials for the studied system.

Name

Type

Min, Max

NPoints

AtomTypes - NB interactions only

MolType - For bonds only

BondNumber - For bonds only

NPairs or NTriplets - For bonds only

Pairs or Triplets - For bonds only

&Table...&EndTable

Note that &Fixed, &InitZero and &InitPMF keywords are not applicable in the included record, but shall be used in the main potential/RDF file instead.